



Armors Labs

BigBang BBCBill Ethereum

Smart Contract Audit

- [BBC Bill Audit Summary](#)
- [BBC Bill Audit](#)
  - [Document information](#)
  - [Review](#)
    - [Audit results](#)
    - [Audited target file](#)
  - [Vulnerability analysis](#)
    - [Vulnerability distribution](#)
    - [Summary of audit results](#)
    - [Contract code](#)
    - [Analysis of audit results](#)
      - [Re-Entrancy](#)
      - [Arithmetic Over/Under Flows](#)
      - [Unexpected Ether](#)
      - [Delegatecall](#)
      - [Default Visibilities](#)
      - [Entropy Illusion](#)
      - [External Contract Referencing](#)
      - [Unsolved TODO comments](#)
      - [Short Address/Parameter Attack](#)
      - [Unchecked CALL Return Values](#)
      - [Race Conditions / Front Running](#)
      - [Denial Of Service \(DOS\)](#)
      - [Block Timestamp Manipulation](#)
      - [Constructors with Care](#)
      - [Unintialised Storage Pointers](#)
      - [Floating Points and Numerical Precision](#)
      - [tx.origin Authentication](#)

# BBC Bill Audit Summary

Project name : BBC Bill for Ethereum.

Project address: None

Code URL : None

Project target : BBC Bill Contract Audit

Test result : PASSED

Audit Info

Audit NO : 0X202101080002

Audit Team : Armors Labs

## BBC Bill Audit

The BBC Group team asked us to review and audit their BBC Bill contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

### Document information

Name	Auditor	Version	Date
BBC Bill Audit	Rock ,Hosea, Rushairer	1.0.0	2021-01-08

### Review

#### Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the BBC Bill contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 18 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

## Audited target file

file	md5
BBCBill_only_erc20.sol	570c62e9ce60f53528e16bdfdf8ed9f0

## Vulnerability analysis

---

### Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

### Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Ether	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe

### Contract code

```

pragma solidity >= 0.5.10;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

interface IERC20Compatible {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

library SafeMath {

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);

```

```

    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold

    return c;
}

function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    return mod(a, b, "SafeMath: modulo by zero");
}

function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b != 0, errorMessage);
    return a % b;
}
}

library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     * - an externally-owned account
     * - a contract in construction
     * - an address where a contract will be created
     * - an address where a contract lived, but was destroyed
     *
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     */

```



```

* https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
*
* IMPORTANT: because control is transferred to `recipient`, care must be
* taken to not create reentrancy vulnerabilities. Consider using
* {ReentrancyGuard} or the
* https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects
*
* _Available since v2.4.0._
*/
function sendValue(address payable recipient, uint256 amount) internal {
    require(address(this).balance >= amount, "Address: insufficient balance");

    // solhint-disable-next-line avoid-call-value
    (bool success, ) = recipient.call.value(amount)("");
    require(success, "Address: unable to send value, recipient may have reverted");
}

}

library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decrease allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing
     * on the return value: the return value is optional (but if data is returned, it must not be false)
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        // 1. The target address is checked to verify it contains contract code
        // 2. The call itself is made, and success asserted
        // 3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
    }
}

```

```

// require(address(token).isContract(), "SafeERC20: call to non-contract");

// solhint-disable-next-line avoid-low-level-calls
(bool success, bytes memory returndata) = address(token).call(data);
require(success, "SafeERC20: low-level call failed");

if (returndata.length > 0) { // Return data is optional
    // solhint-disable-next-line max-line-length
    require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
}
}
}

library SafeERC20Compatible {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20Compatible token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20Compatible token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20Compatible token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20Compatible token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20Compatible token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decrease allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing
     * the return value: the return value is optional (but if data is returned, it must not be false)
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20Compatible token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        // 1. The target address is checked to verify it contains contract code
        // 2. The call itself is made, and success asserted
        // 3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        // require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");
    }
}

```



```

        // if (returndata.length > 0) { // Return data is optional
        //     // solhint-disable-next-line max-line-length
        //     require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        // }
    }
}

contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;
    address private _admin;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
    event AdminTransferred(address indexed previousOwner, address indexed newAdmin);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        _admin = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    function admin() public view returns (address) {
        return _admin;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
    modifier onlyAdmin() {
        require(isAdmin(), "Ownable: caller is not the admin");
        _;
    }
}

/**
 * @dev Returns true if the caller is the current owner.

```

```

    */
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }

    function isAdmin() public view returns (bool) {
        return _msgSender() == _admin;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    function transferAdminship(address newAdmin) public onlyOwner {
        require(newAdmin != address(0), "Ownable: new admin is the zero address");
        emit AdminTransferred(_owner, newAdmin);
        _admin = newAdmin;
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract BBCBill is Ownable {

    using SafeMath for uint256;
    using SafeERC20 for IERC20;
    using SafeERC20Compatible for IERC20Compatible;
    mapping(address => bool) public _compatible;
    // event
    event debug(uint256 msg);
    event inflowEvent(address owner, uint256 amount, address indexed contractAddress,
        string bbcBranch, string bbcAddress);
    event retraceEvent(uint256 txid, uint256 amount, address walletAddress, address contractAddress,
        uint256 fee);
    event withdrawEvent(string txid, string bbcBranch, address walletAddress,
        address contractAddress, uint256 amount, uint256 fee, uint256 rate);

    function compatible(address sellToken, bool isCompatible) public onlyOwner returns (bool) {
        _compatible[sellToken] = isCompatible;
        return true;
    }

    function inflow(uint256 _amount, address _contractAddress,

```

```

        string memory _bbcBranch, string memory _bbcAddress) public payable{
    require(_amount > 0, "inFlow:_amount invalid");
    require(_contractAddress != address(0), "inFlow:contract invalid");
    if(_compatible[_contractAddress]){
        IERC20Compatible(_contractAddress).safeTransferFrom(msg.sender, address(this), _amount);
    }else{
        IERC20(_contractAddress).safeTransferFrom(msg.sender, address(this), _amount);
    }
    emit inflowEvent(msg.sender, _amount, _contractAddress, _bbcBranch, _bbcAddress);
}

function retrace(uint256 _txid, uint256 _amount, address _walletAddress, address _contractAddress
uint256 _fee) public onlyAdmin{
    require(_amount > 0, "retrace:_amount invalid");
    require(_contractAddress != address(0), "retrace:contract invalid");
    require(_walletAddress != address(0), "retrace:walletAddress invalid");
    _withdraw(_contractAddress, _walletAddress, _amount, _fee);
    emit retraceEvent(_txid, _amount, _walletAddress, _contractAddress, _fee);
}

function withdraw(string memory _txid, string memory _bbcBranch, address _walletAddress,
address _contractAddress, uint256 _amount, uint256 _fee, uint256 _rate) public onlyAdmin{
    require(_amount > 0, "withdraw:_amount invalid");
    require(_contractAddress != address(0), "withdraw:contract invalid");
    require(_walletAddress != address(0), "withdraw:walletAddress invalid");
    _withdraw(_contractAddress, _walletAddress, _amount, _fee);
    emit withdrawEvent(_txid, _bbcBranch, _walletAddress, _contractAddress, _amount, _fee, _rate)
}

function _withdraw(address _contractAddress, address _walletAddress,
uint256 _amount, uint256 _fee) internal onlyAdmin{
    if(_compatible[_contractAddress]){
        IERC20Compatible(_contractAddress).safeTransfer(_walletAddress, _amount);
        IERC20Compatible(_contractAddress).safeTransfer(admin(), _fee);
    }else{
        IERC20(_contractAddress).safeTransfer(_walletAddress, _amount);
        IERC20(_contractAddress).safeTransfer(admin(), _fee);
    }
}
}

```

## Analysis of audit results

### Re-Entrancy

- Description:**

One of the features of Ethereum smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle ether, and as such often send ether to various external user addresses. The operation of calling external contracts, or sending ether to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function), including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.

- Detection results:**

PASSED!

- Security suggestion:**

no.

## Arithmetic Over/Under Flows

- **Description:**

The Ethereum Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Unexpected Ether

- **Description:**

Typically when ether is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where ether can exist in a contract without having executed any code. Contracts which rely on code execution for every ether sent to the contract can be vulnerable to attacks where ether is forcibly sent to a contract.

- **Detection results:**

PASSED!

- **Security suggestion:** no.

## Delegatecall

- **Description:**

The CALL and DELEGATECALL opcodes are useful in allowing Ethereum developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.

- **Detection results:**

PASSED!

- **Security suggestion:** no.

## Default Visibilities

- **Description:**

Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions

default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devastating vulnerabilities in smart contracts as will be discussed in this section.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Entropy Illusion

---

- **Description:**

All transactions on the Ethereum blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the Ethereum ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no `rand()` function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## External Contract Referencing

---

- **Description:**

One of the benefits of the Ethereum global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Unsolved TODO comments

---

- **Description:**

Check for Unsolved TODO comments

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Short Address/Parameter Attack

- **Description:**

This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Unchecked CALL Return Values

- **Description:**

There are a number of ways of performing external calls in solidity. Sending ether to external accounts is commonly performed via the `transfer()` method. However, the `send()` function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The `call()` and `send()` functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (initialised by `call()` or `send()`) fails, rather the `call()` or `send()` will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Race Conditions / Front Running

- **Description:**

The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the Ethereum blockchain. There is a variety of good posts on this subject, a few are: [Ethereum Wiki - Safety](#), [DASP - Front-Running](#) and the [Consensus - Smart Contract Best Practices](#).

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## Denial Of Service (DOS)

- **Description:**

This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap ether in these contracts forever, as was the case with the Second Parity MultiSig hack

- **Detection results:**



PASSED!

- **Security suggestion:**  
no.

## Block Timestamp Manipulation

---

- **Description:**

Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.

- **Detection results:**

PASSED!

- **Security suggestion:**  
no.

## Constructors with Care

---

- **Description:**

Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.

- **Detection results:**

PASSED!

- **Security suggestion:**  
no.

## Unintialised Storage Pointers

---

- **Description:**

The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately initialising variables.

- **Detection results:**

PASSED!

- **Security suggestion:**  
no.

## Floating Points and Numerical Precision

---

- **Description:**

As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

## tx.origin Authentication

---

- **Description:**

Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.

- **Detection results:**

PASSED!

- **Security suggestion:**

no.

Armors Labs



armors.io

contact@armors.io

